Building Java Programs

Chapter 3:
Introduction to Parameters and Objects

Chapter outline

Lecture 6

- **parameters**
  - **passing parameters to static methods**
  - **writing methods that accept parameters**
- **methods that return values**
  - **calling methods that return values (e.g. the `Math` class)**
  - **writing methods that return values**

Lecture 7

- type casting
- using objects
- console input with Scanner objects
- cumulative sum

Parameters

- suggested reading: 3.1

Reminder: global constants

- In the last chapter, we used global constants to fix "magic number" redundancy problems:

```
public static final int FIGURE_WIDTH = 5;

public static void drawFigure1() {
    drawPlusLine();
    drawBarLine();
    drawPlusLine();
}

public static void drawPlusLine() {
    System.out.print("+");
    for (int i = 1; i <= FIGURE_WIDTH; i++) {
        System.out.print("/\\");
    }
    System.out.println("+");
}

public static void drawBarLine() {
    System.out.print("|");
    for (int i = 1; i <= 2 * FIGURE_WIDTH; i++) {
        System.out.print(" ");
    }
    System.out.println("|");
}
```

Another repetitive figure

- Now consider the task of drawing the following figures:

```
**********

*******

**********

**********
*        *
**********

*****
*   *
*   *
*****
```

  - The lines and figures are similar, but not exactly the same.

A redundant solution

```
public class Stars1 {
    public static void main(String[] args) {
        drawLineOf13Stars();
        drawLineOf7Stars();
        drawLineOf35Stars();
        draw10x3Box();
        draw5x4Box();
    }

    public static void drawLineOf13Stars() {
        for (int i = 1; i <= 13; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void drawLineOf7Stars() {
        for (int i = 1; i <= 7; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void drawLineOf35Stars() {
        for (int i = 1; i <= 35; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
    ...
```

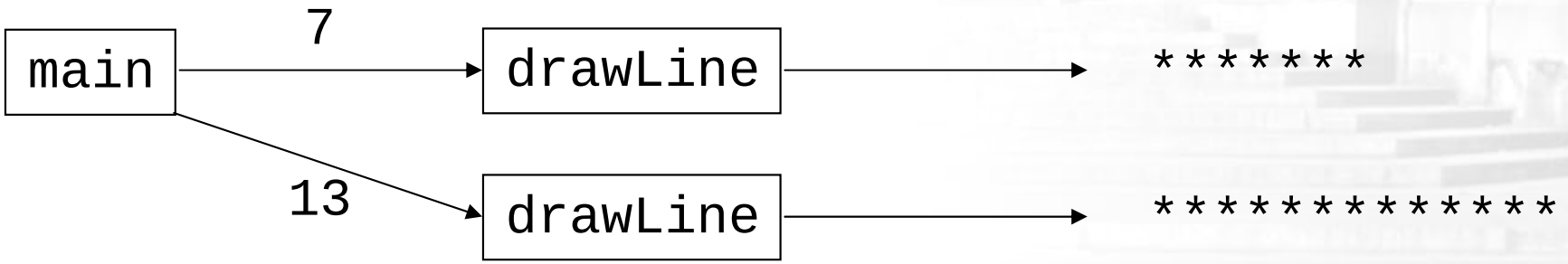
- The methods at left are redundant.
- Would constants help us solve this problem?
- What would be a better solution?
  - `drawLine` - A method to draw a line of any number of stars.
  - `drawBox` - A method to draw a box of any size.

Parameterization

- **parameterized method**: One that is given extra information (e.g. number of stars to draw) when it is called.

- **parameter**: A value passed to a method by its caller.

- Writing parameterized methods requires 2 steps:
  - *write* the method to accept the parameter
  - *call* the method and pass the parameter value(s) desired



Writing parameterized methods

- Parameterized method declaration syntax:

```
public static void <name> ( <type> <name> ) {
    <statement(s)> ;
}
```

- Example:

```
public static void printSpaces(int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(" ");
    }
}
```

  - Whenever `printSpaces` is called, the caller must specify how many spaces to print.

Calling parameterized methods

- **passing a parameter**: Calling a parameterized method and specifying a value for its parameter(s).

- Parameterized method call syntax:

```
<name> ( <expression> );
```

- Example:

```
System.out.print("*");
printSpaces(7);
System.out.print("**");
int x = 3 * 5;
printSpaces(x + 2);
System.out.println("***");
```

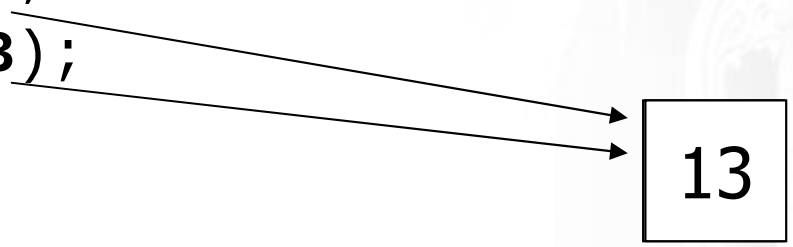
  - Output:

```
*       **                      ***
```

How parameters are passed

- When the parameterized method call executes:
  - the value written is copied into the parameter variable
  - the method's code executes using that value

```
public static void main(String[] args) {
    printSpaces(7);
    printSpaces(13);
}

public static void printSpaces(int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(" ");
    }
}
```



Value semantics

- **value semantics**: When primitive variables (such as `int` or `double`) are passed as parameters in Java, their values are copied.
  - Modifying the parameter inside the method will not affect the variable passed in.

```
public static void main(String[] args) {
    int x = 23;
    strange(x);
    System.out.println("2. x = " + x);    // unchanged
    ...
}

public static void strange(int x) {
    x = x + 1;
    System.out.println("1. x = " + x);
}
```

Output:

```
1. x = 24
2. x = 23
```

Common errors

- If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.

```
printSpaces();   // ERROR: parameter value required
```

- The value passed to a method must be of the correct type, matching the type of its parameter variable.

```
printSpaces(3.7);   // ERROR: must be of type int
```

- Exercise: Change the Stars program to use parameterized static methods.

Stars solution

```
// Prints several lines of stars.
// Uses a parameterized method to remove redundancy.

public class Stars2 {
    public static void main(String[] args) {
        drawLine(13);
        drawLine(7);
        drawLine(35);
    }

    // Prints the given number of stars plus a line break.
    public static void drawLine(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
```

Multiple parameters

- Methods can accept as many parameters as you like.
  - The parameters are separated by commas.
  - When the method is called, it must be passed values for each of its parameters.

- Multiple parameters declaration syntax:

```
public static void <name> ( <type> <name> ,
        <type> <name> , ..., <type> <name> ) {
    <statement(s)> ;
}
```

- Multiple parameters call syntax:

```
<name> ( <expression>, <expression>, ..., <expression> );
```

Multiple parameters example

```
public static void main(String[] args) {
    printNumber(4, 9);
    printNumber(17, 6);
    printNumber(8, 0);
    printNumber(0, 8);
}

public static void printNumber(int number, int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(number);
    }
    System.out.println();
}
```

Output:

```
444444444
171717171717

00000000
```

- Exercise: Write an improved version of the Stars program that draws its boxes of stars using parameterized static methods.

Stars solution

```
// Prints several lines and boxes made of stars.
// Third version with multiple parameterized methods.

public class Stars3 {
    public static void main(String[] args) {
        drawLine(13);
        drawLine(7);
        drawLine(35);
        System.out.println();
        drawBox(10, 3);
        drawBox(5, 4);
        drawBox(20, 7);
    }

    // Prints the given number of stars plus a line break.
    public static void drawLine(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
```

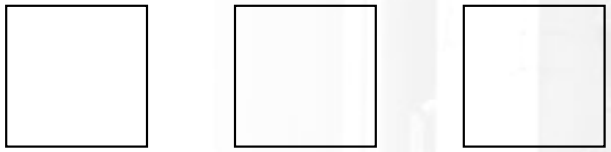
Stars solution, cont'd.

```
    // Prints a box of stars of the given size.
    public static void drawBox(int width, int height) {
        drawLine(width);

        for (int i = 1; i <= height - 2; i++) {
            System.out.print("*");
            printSpaces(width - 2);
            System.out.println("*");
        }

        drawLine(width);
    }

    // Prints the given number of spaces.
    public static void printSpaces(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print(" ");
        }
    }
}
```

Parameter "mystery" problem

- What is the output of the following program?

```
public class Mystery {
    public static void main(String[] args) {
        int x = 5, y = 9, z = 2;
        mystery(z, y, x);
        System.out.println(x + " " + y + " " + z);
        mystery(y, x, z);
        System.out.println(x + " " + y + " " + z);
    }

    public static void mystery(int x, int z, int y) {
        x++;
        y = x - z * 2;
        x = z + 1;
        System.out.println(x + " " + y + " " + z);
    }
}
```



Parameter questions

- Rewrite the following program to use parameterized methods:

```
// Draws triangular figures of stars.

public class Loops {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i - 1; j++) {
                System.out.print(" ");
            }
            for (int j = 1; j <= 10 - 2 * i + 1; j++) {
                System.out.print("*");
            }
            System.out.println();
        }

        for (int i = 1; i <= 12; i++) {
            for (int j = 1; j <= i - 1; j++) {
                System.out.print(" ");
            }
            for (int j = 1; j <= 25 - 2 * i; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

Parameter answer

- Rewrite the following program to use parameterized methods:

```
// Draws triangular figures using parameterized methods.

public class Loops {
    public static void main(String[] args) {
        triangle(5);
        triangle(12);
    }

    // Draws a triangle figure of the given size.
    public static void triangle(int height) {
        for (int i = 1; i <= height; i++) {
            printSpaces(i - 1);
            drawLine(2 * height + 1 - 2 * i);
        }
    }

    ...
}
```

Parameter questions

- Write a method named `printDiamond` that accepts a height as a parameter and prints a diamond figure:

```
  *
 ***
*****
 ***
  *
```

- Write a method named `multiplicationTable` that accepts a maximum integer as a parameter and prints a table of multiplication from `1 x 1` up to that integer times itself.
- Write a method named `bottlesOfBeer` that accepts an integer as a parameter and prints the "XX Bottles of Beer" song with that many verses.

Methods that return values

- suggested reading: 3.2

Java's `Math` class

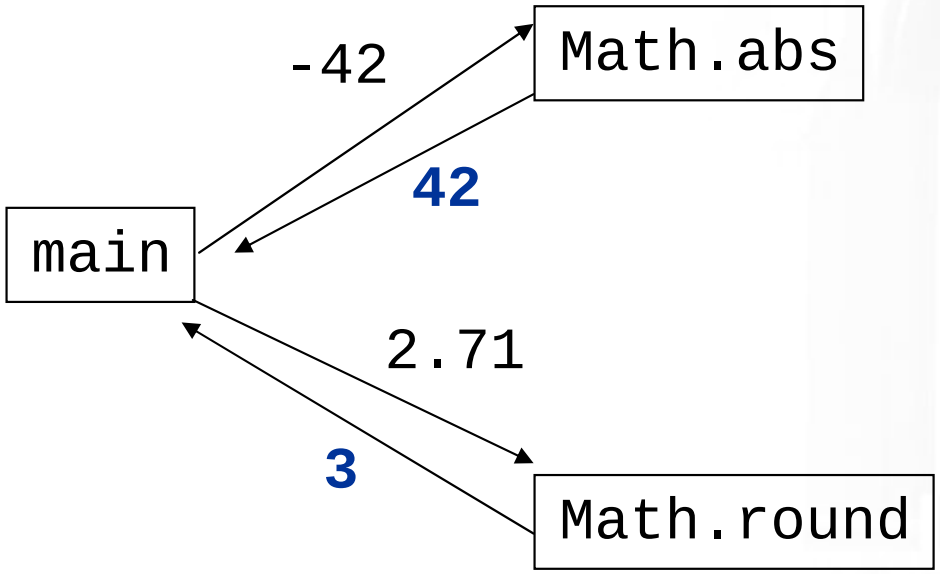
- Java has a class called `Math` that has several useful static methods and constants for performing mathematical calculations.

Method name	Description
abs(*value*)	absolute value
ceil(*value*)	rounds up
cos(*value*)	cosine, in radians
floor(*value*)	rounds down
log(*value*)	logarithm base *e*
log10(*value*)	logarithm base 10
max(*value1*, *value2*)	larger of two values
min(*value1*, *value2*)	smaller of two values
pow(*base*, *exponent*)	*base* to the *exponent* power
random()	random `double` between 0 and 1
round(*value*)	nearest whole number
sin(*value*)	sine, in radians
sqrt(*value*)	square root

Constant	Description
E	2.7182818...
PI	3.1415926...

Methods that return values

- **return**: To send a value out as the result of a method, which can be used in an expression.
  - A return is like the opposite of a parameter:
    - Parameters pass information *in* from the caller to the method.
    - Return values pass information *out* from a method to its caller.



  - The `Math` methods do not print results to the console. Instead, each method evaluates to produce (or *return*) a numeric result, which can be used in an expression.

Math method examples

- `Math` method call syntax:

```
Math.<method name>(<parameter(s)>)
```

- Examples:

```
double squareRoot = Math.sqrt(121.0);
System.out.println(squareRoot);              // 11.0

int absoluteValue = Math.abs(-50);
System.out.println(absoluteValue);           // 50

System.out.println(Math.min(3, 7) + 2);      // 5
```

  - Notice that the preceding calls are used in expressions; they can be printed, stored into a variable, etc.

Math method questions

- Evaluate the following expressions:
  - `Math.abs(-1.23)`
  - `Math.pow(3, 2)`
  - `Math.pow(10, -2)`
  - `Math.sqrt(121.0) - Math.sqrt(256.0)`
  - `Math.round(Math.PI) + Math.round(Math.E)`
  - `Math.ceil(6.022) + Math.floor(15.9994)`
  - `Math.abs(Math.min(-3, -5))`

- `Math.max` and `Math.min` can be used to bound numbers.

  Consider an `int` variable named `age`.
  - What statement would replace negative ages with 0?
  - What statement would cap the maximum age to 40?

Methods that return values

- Syntax for methods that return a value:

```
public static <type> <name> ( < parameter(s)> ) {
    < statement(s)> ;
}
```

  - Returning a value from a method:

```
    return <expression> ;
```

- Example:

```
// Returns the slope of the line between the given points.
public static double slope(int x1, int y1, int x2, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    return dy / dx ;
}
```

Return examples

```
// Converts Fahrenheit to Celsius.
public static double fToC(double degreesF) {
    double degreesC = 5.0 / 9.0 * (degreesF - 32);
    return degreesC;
}

// Computes length of triangle hypotenuse given its side lengths.
public static double hypotenuse(int a, int b) {
    double c = Math.sqrt(a * a + b * b);
    return c;
}

// Rounds the given number to two decimal places.
// Example: round(2.71828183) returns 2.72.
public static double round2(double value) {
    double result = value * 100;    // upscale the number
    result = Math.round(result);    // round to nearest integer
    result = result / 100;          // downscale the number
    return result;
}
```

Return examples shortened

```
// Converts Fahrenheit to Celsius.
public static double fToC(double degreesF) {
    return 5.0 / 9.0 * (degreesF - 32);
}


// Computes length of triangle hypotenuse given its side lengths.
public static double hypotenuse(int a, int b) {
    return Math.sqrt(a * a + b * b);
}


// Rounds the given number to two decimal places.
// Example: round(2.71828183) returns 2.72.
public static double round2(double value) {
    return Math.round(value * 100) / 100;
}
```

Return questions

- Write a method named `area` that accepts a circle's radius as a parameter and returns its area.
  - You may wish to use the constant `Math.PI` in your solution.

- Write a method named `distanceFromOrigin` that accepts x and y coordinates as parameters and returns the distance between that (x, y) point and the origin.

- Write a method named `attendance` that accepts a number of lectures attended by a student, and returns how many points a student receives for attendance. The student receives 2 points for each of the first 5 lectures and 1 point for each subsequent lecture.

How to comment: methods 2

- If your method accepts parameters and/or returns a value, write a brief description of what the parameters are used for and what kind of value will be returned.
  - In your comments, you can also write your assumptions about the values of the parameters.
  - You may wish to give examples of what values your method returns for various input parameter values.

  - Example:

```
// This method returns the factorial of the given integer n.
// The factorial is the product of all integers up to that number.
// Assumes that the parameter value is non-negative.
// Example: factorial(5) returns 1 * 2 * 3 * 4 * 5 = 120.
public static int factorial(int n) {
    ...
}
```

Chapter outline

Lecture 6

- parameters
  - passing parameters to static methods
  - writing methods that accept parameters
- methods that return values
  - calling methods that return values (e.g. the `Math` class)
  - writing methods that return values

Lecture 7

- **type casting**
- **using objects**
- **console input with Scanner objects**
- **cumulative sum**

Type casting

- **type cast**: A conversion from one type to another.

  Common uses:
  - To promote an `int` into a `double` to achieve exact division.
  - To truncate a `double` from a real number to an integer.

- type cast general syntax:

  (***<type>***) ***<expression>***

  Examples:
  - `double result = (double) 19 / 5;  // 3.8`
  - `int result2 = (int) result;  // 3`

More about type casting

- Type casting has high precedence and only casts the item immediately next to it.
  - `double x = (double) 1 + 1 / 2;  // 1`
  - `double y = 1 + (double) 1 / 2;  // 1.5`
- You can use parentheses to force evaluation order.
  - `double average = (double) (a + b + c) / 3;`
- A conversion to `double` can be achieved in other ways.
  - `double average = 1.0 * (a + b + c) / 3;`

Using objects

- suggested reading: 3.3

Objects

- So far, we have seen:
  - **methods**, which represent behavior
  - **variables**, which represent data (categorized by **types**)

- It is possible to create new types that are combinations of the existing types.
  - Such types are called *object types* or *reference types*.
  - Languages such as Java in which you can do this are called *object-oriented* programming languages.

- We will learn how to use some of Java's objects.
  - In Chapter 8 we will learn to create our own types of objects.

Objects and classes

- **object:** An entity that contains data and behavior.
  - There are variables inside the object, representing its data.
  - There are methods inside the object, representing its behavior.

- **class**:
  - A program, or...
  - A type of objects.

- Examples:
  - The class `String` represents objects that store text characters.
  - The class `Point` represents objects that store (x, y) data.
  - The class `Scanner` represents objects that read information from the keyboard, files, and other sources.

Constructing objects

- **construct**: To create a new object.
  - Objects are constructed with the `new` keyword.
  - Most objects must be constructed before they can be used.
- Constructing objects, general syntax:

```
<type> <name> = new <type> ( <parameters> );
```

  - Examples:

```
Point p = new Point(7, -4);
DrawingPanel window = new DrawingPanel(300, 200);
Color orange = new Color(255, 128, 0);
```

  - Classes' names are usually uppercase (e.g. `Point`, `Color`).
  - Strings are also objects, but are constructed without `new` :

```
String name = "Amanda Ann Camp";
```

Calling methods of objects

- Objects contain methods that can be called by your program.
  - For example, a `String`'s methods manipulate or process the text of that `String` in useful ways.
  - When we call an object's method, we are sending a message to it.
  - We must specify which object we are talking to, and then write the method's name.

- Calling a method of an object, general syntax:
  <variable> . ***<method name>*** (***<parameters>***)
  - The results will be different from one object to another.
  - Examples:

```
String gangsta = "G., Ali";
System.out.println(gangsta.length());    // 7

Point p1 = new Point(3, 4);
Point p2 = new Point(0, 0);
System.out.println(p1.distance(p2));     // 5.0
```

Interactive programs using Scanner objects

- suggested reading: 3.4

Interactive programs

- We have written programs that print console output.
- It is also possible to read *input* from the console.
  - The user types the input into the console.
  - We can capture the input and use it in our program.
  - Such a program is called an *interactive program*.
- Interactive programs can be challenging:
  - Computers and users think in very different ways.
  - Users tend to misbehave.

Input and System.in

- We print output using an object named **System.out**
  - This object has methods named `println` and `print`.
- We read input using an object named **System.in**
  - `System.in` is not intended to be used directly.
  - We will use a second object, from a class called `Scanner`, to help us read input from `System.in`.
- Constructing a `Scanner` object to read console input:

  ```
  Scanner <name> = new Scanner(System.in);
  ```

  - Example:

    ```
    Scanner console = new Scanner(System.in);
    ```

  - Once we have constructed the `Scanner`, we call various methods on it to read the input from the user.

Scanner methods

- Methods of `Scanner` that we will use in this chapter:

Method	Description
`nextInt()`	reads and returns user input as an `int`
`nextDouble()`	reads and returns user input as a `double`
`next()`	reads and returns user input as a `String`

  - Each of these methods pauses your program until the user types input and presses Enter.
    - Then the value typed is *returned* to your program.
  - **prompt**: A message printed to the user, telling them what input to type, before we read from the `Scanner`.

- Example:

```
System.out.print("How old are you? ");    // prompt
int age = console.nextInt();
System.out.println("You'll be 40 in " + (40 - age)
                   + " years.");
```

Java class libraries, import

- **Java class libraries**: A large set of Java classes available for you to use (part of the JDK).
  - These objects are organized into groups named *packages*.
  - To use the objects from a package, you must include an *import declaration* at the top of your program.
  - `Scanner` is in a package named `java.util`

- Import declaration, general syntax:

```
import <package name> .*;
```

- To use `Scanner`, put this at the start of your program:

```
import java.util.*;
```

Input tokens

- **token**: A unit of user input, as read by the `Scanner`.
  - Tokens are separated by whitespace (spaces, tabs, new lines).
  - How many tokens appear on the following line of input?

```
23  John Smith   42.0 "Hello world"
```

- When the token doesn't match the type the `Scanner` tries to read, the program crashes.

Example:

```
System.out.print("What is your age? ");
int age = console.nextInt();
```

Output (user's input is underlined):

```
What is your age? Timmy
java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        ...
```

Example Scanner usage

```
import java.util.*;   // so that I can use Scanner

public class ReadSomeInput {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("What is your first name? ");
        String name = console.next();

        System.out.print("And how old are you? ");
        int age = console.nextInt();

        System.out.println(name + " is " + age);
        System.out.println("That's quite old!");
    }
}
```

- Output (user input underlined):

```
What is your first name? Ruth
How old are you? 14
Ruth is 14
That's quite old!
```

Another Scanner example

```
import java.util.*;   // so that I can use Scanner

public class Average {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Please type three numbers: ");
        int num1 = console.nextInt();
        int num2 = console.nextInt();
        int num3 = console.nextInt();

        double average = (double) (num1 + num2 + num3) / 3;
        System.out.println("The average is " + average);
    }
}
```

- Output (user input underlined):

```
Please type three numbers: 8 6 13
The average is 9.0
```

  - Notice that the Scanner can read multiple values from one line.

Scanners as parameters

- If multiple methods read user input, declare a Scanner in main and pass it to each of them as a parameter.
  - In this way, all of the methods share the same Scanner object.

```
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    int sum = readSum3(console);
    System.out.println("The sum is " + sum);
}

public static int readSum3(Scanner console) {
    System.out.print("Type 3 numbers: ");
    int num1 = console.nextInt();
    int num2 = console.nextInt();
    int num3 = console.nextInt();
    return num1 + num2 + num3;
}
```

Scanner BMI question

A person's body mass index (BMI) is computed by the following formula:

$$BMI = \frac{weight}{height^2} \times 703$$

- Write a program that produces the following output:

```
This program reads in data for two people
and computes their body mass index (BMI)
and weight status.

Enter next person's information:
height (in inches)? 62.5
weight (in pounds)? 130.5

Enter next person's information:
height (in inches)? 58.5
weight (in pounds)? 90

Person #1 body mass index = 23.485824
Person #2 body mass index = 18.487836949375414
Difference = 4.997987050624587
```

Scanner BMI solution

```
// This program computes two people's body mass index (BMI)
// and compares them.  The code uses parameters and returns.

import java.util.*;  // so that I can use Scanner

public class BMI {
    public static void main(String[] args) {
        introduction();
        Scanner console = new Scanner(System.in);

        double bmi1 = processPerson(console);
        double bmi2 = processPerson(console);

        // report overall results
        System.out.println("Person #1 body mass index = " + bmi1);
        System.out.println("Person #2 body mass index = " + bmi2);
        double difference = Math.abs(bmi1 - bmi2);
        System.out.println("Difference = " + difference);
    }

    // prints a welcome message explaining the program
    public static void introduction() {
        System.out.println("This program reads in data for two people");
        System.out.println("and computes their body mass index (BMI)");
        System.out.println("and weight status.");
        System.out.println();
    }
    ...
```

Scanner BMI solution, cont.

```
    ...

    // reads information for one person, computes their BMI, and returns it
    public static double processPerson(Scanner console) {
        System.out.println("Enter next person's information:");
        System.out.print("height (in inches)? ");
        double height = console.nextDouble();

        System.out.print("weight (in pounds)? ");
        double weight = console.nextDouble();
        System.out.println();

        double bmi = getBMI(height, weight);
        return bmi;
    }

    // Computes a person's body mass index based on their height and weight
    // and returns the BMI as its result.
    public static double getBMI(double height, double weight) {
        double bmi = weight / (height * height) * 703;
        return bmi;
    }
}
```

Cumulative sum

- suggested reading: 4.1

Adding many numbers

- Consider the following code to read three values from the user and add them together:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int num1 = console.nextInt();

System.out.print("Type a number: ");
int num2 = console.nextInt();

System.out.print("Type a number: ");
int num3 = console.nextInt();

int sum = num1 + num2 + num3;

System.out.println("The sum is " + sum);
```

A cumulative sum

- The variables num1, num2, and num3 are unnecessary:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int sum = console.nextInt();

System.out.print("Type a number: ");
sum += console.nextInt();

System.out.print("Type a number: ");
sum += console.nextInt();

System.out.println("The sum is " + sum);
```

- **cumulative sum**: A variable that keeps a sum-in-progress and is updated many times until the task of summing is finished.
  - The variable sum in the above code represents a cumulative sum.

Failed cumulative sum loop

- How could we modify the code to sum 100 numbers?
  - Creating 100 copies of the same code would be redundant.

- An incorrect solution:
  - The scope of `sum` is inside the `for` loop, so the last line of code fails to compile.

```
Scanner console = new Scanner(System.in);
for (int i = 1; i <= 100; i++) {
    int sum = 0;
    System.out.print("Type a number: ");
    sum += console.nextInt();
}

// sum is undefined here
System.out.println("The sum is " + sum);
```

Fixed cumulative sum loop

- A corrected version of the sum loop code:

```
Scanner console = new Scanner(System.in);
int sum = 0;
for (int i = 1; i <= 100; i++) {
    System.out.print("Type a number: ");
    sum += console.nextInt();
}
System.out.println("The sum is " + sum);
```

The key idea:

- Cumulative sum variables must always be declared outside the loops that update them, so that they will continue to live after the loop is finished.

User-guided cumulative sum

- The user's input can control the number of times the loop repeats:

```
Scanner console = new Scanner(System.in);
System.out.print("How many numbers to add? ");
int count = console.nextInt();

int sum = 0;
for (int i = 1; i <= count; i++) {
    System.out.print("Type a number: ");
    sum += console.nextInt();
}
System.out.println("The sum is " + sum);
```

  - An example output:

```
How many numbers to add? 3
Type a number: 2
Type a number: 6
Type a number: 3
The sum is 11
```

Variation: cumulative product

- The same idea can be used with other operators, such as multiplication which produces a cumulative product:

```
Scanner console = new Scanner(System.in);
System.out.print("Raise 2 to what power? ");
int exponent = console.nextInt();

int product = 1;
for (int i = 1; i <= exponent; i++) {
    product = product * 2;
}
System.out.println("2 to the " + exponent + " = " + product);
```

- Possible exercises:
  - Change the above code so that it also prompts for the base, instead of always using 2.
  - Change the above code into a method which accepts a base *a* and exponent *b* as parameters and returns a^b.

Cumulative sum question

- Write a program that reads input of the number of hours two employees have worked and displays each employee's total and the overall total hours.
  - The company doesn't pay overtime, so cap any day at 8 hours.
- Example log of execution:

```
Employee 1: How many days? 3
Hours? 6
Hours? 12
Hours? 5
Employee 1's total paid hours = 19

Employee 2: How many days? 2
Hours? 11
Hours? 6
Employee 2's total hours = 14

Total paid hours for both employees = 33
```

Cumulative sum solution

```
// Computes the total paid hours worked by two employees.
// The company does not pay for more than 8 hours per day.
// Uses a "cumulative sum" loop to compute the total hours.

import java.util.*;

public class Hours {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        int hours1 = processEmployee(console, 1);
        int hours2 = processEmployee(console, 2);

        int total = hours1 + hours2;
        System.out.println("Total paid hours for both employees = " + total);
    }

    ...
```

Cumulative sum solution 2

```
    ...

    // Reads hours information about one employee with the given number.
    // Returns the total hours worked by the employee.
    public static int processEmployee(Scanner console, int number) {
        System.out.print("Employee " + number + ": How many days? ");
        int days = console.nextInt();

        // totalHours is a cumulative sum of all days' hours worked.
        int totalHours = 0;
        for (int i = 1; i <= days; i++) {
            System.out.print("Hours? ");
            int hours = console.nextInt();
            hours = Math.min(hours, 8);       // cap at 8 hours per day
            totalHours = totalHours + hours;
        }

        System.out.println("Employee " + number + "'s total paid hours = "
                           + totalHours);
        System.out.println();
        return totalHours;
    }
}
```